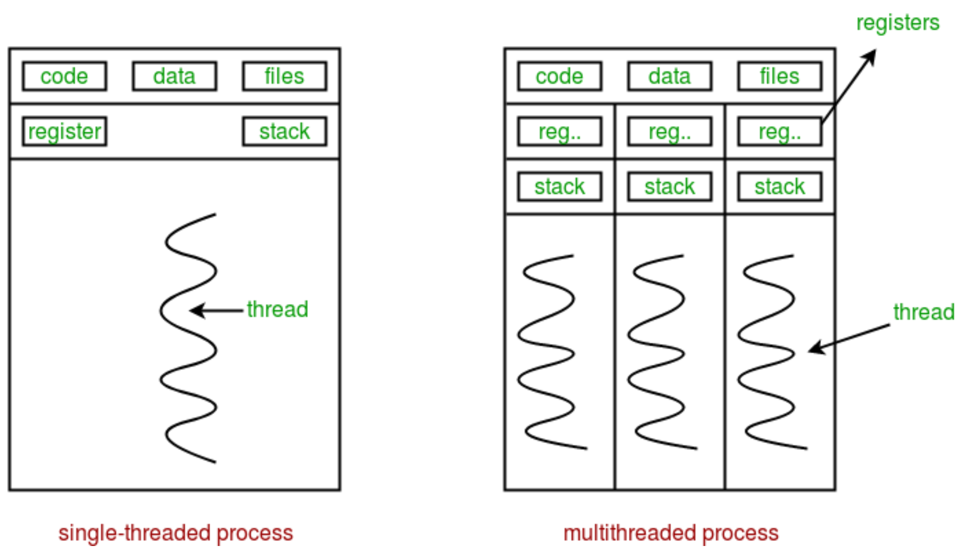
MultiThreaded app debug with NetBeans



1. Debugging multithreaded applications

Debugging applications is one of the tasks that we programmers perform most often. It allows us to verify the status of our application at a specific point. Putting marks in the code or using the debugging tools (debugging, in English) with all the utilities that some IDEs provide us, is an essential task to review our application. Debugging a sequential application where there is only one line of execution is simple with a debugging tool. We are advancing step by step to the thread of execution and we are looking at variable values, conditions... until we finish. But in multithreaded programming there is more than one thread of execution. We will have to consult each thread separately to see how it works and what values it is taking.

We will see how we can debug a multi-threaded application and use the tool to detect possible errors such as deadlock in an application.

To perform the tests we will use the Netbeans IDE. With other IDEs the operation is similar. To test how to query the threads we will open a project and select the project in the projects window and choose the Debug option. The project that we will analyze is the following:

`Shuttle.java`

```java
package UD02.SpaceShuttle;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import javax.swing.ImageIcon;

class Shuttle {

    private int x, y;
    private int dsx, dsy, v; //displacement, v sleep
    private int tx = 37; //margin x
    private int ty = 63; //margin y image (74x126 pixels)
    private Image image;

    public Shuttle(int x, int y, int dsx, int dsy, int v)  {

        this.x = x;
        this.y = y;
        this.dsx = dsx;
        this.dsy = dsy;
        this.v = v;
        image = new ImageIcon(Shuttle.class.getResource("shuttle.png")).getImage();
    }

    public int velocity() { //sleep in miliseconds
        return v;
    }

    public void move() {
        x = x + dsx;
        y = y + dsy;
        // touch the margin?
        if (x >= (400 - tx*2) || x <= 0) {
            dsx = -dsx;
        }
        if (y >= (600 - ty*2) || y <= 0) {
            dsy = -dsy;
        }
    }

    public void draw(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        g2d.drawImage(this.image, x, y, null);
    }
}
```

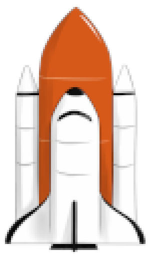
`ShuttlePanel.java`

```java
package UD02.SpaceShuttle;

import java.awt.Graphics;
import java.util.Random;
import javax.swing.JPanel;

class ShuttlePanel extends JPanel implements Runnable {

    private int shuttleNumber = 3;
    Thread[] shuttleThreads;
    Shuttle[] shuttle;

    public ShuttlePanel() {
        shuttleThreads = new Thread[shuttleNumber];
        shuttle = new Shuttle[shuttleNumber];
        for (int i = 0; i < shuttleThreads.length; i++) {
            shuttleThreads[i] = new Thread(this);
            shuttleThreads[i].setName("Shuttle Thread - " + i);
            Random rand = new Random();
            int velocity = rand.nextInt(50);
            int posX = rand.nextInt(100) + 30;
            int posY = rand.nextInt(100) + 30;
            int dX = rand.nextInt(3) + 1;
            int dY = rand.nextInt(3) + 1;
            shuttle[i] = new Shuttle(posX, posY, dX, dY, velocity);
        }
        for (int i = 0; i < shuttleThreads.length; ++i) {
            shuttleThreads[i].start();
        }
    }

    @Override
    public void run() {
        for (int i = 0; i < shuttleThreads.length; ++i) {
            while (shuttleThreads[i].currentThread() == shuttleThreads[i]) {
                try {
                    shuttleThreads[i].sleep(shuttle[i].velocity()); //
                    shuttle[i].move();
                } catch (InterruptedException e) {
                }
                repaint();
            }
        }
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        for (int i = 0; i < shuttle.length; ++i) {
            shuttle[i].draw(g);
        }
    }
}
```

`SpaceShuttle.java`

```java
package UD02.SpaceShuttle;

import javax.swing.JFrame;

public class SpaceShuttle extends javax.swing.JFrame {

    public SpaceShuttle() {
        initComponents();
    }

    @SuppressWarnings("unchecked")
    private void initComponents() {
        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
    }

    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info : 
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {
            java.util.logging.Logger.getLogger(SpaceShuttle.class.getName()).log(
                    java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {
            java.util.logging.Logger.getLogger(SpaceShuttle.class.getName()).log(
                    java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(SpaceShuttle.class.getName()).log(
                    java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(SpaceShuttle.class.getName()).log(
                    java.util.logging.Level.SEVERE, null, ex);
        }

        SpaceShuttle f = new SpaceShuttle();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setTitle("Space Shuttle");
        f.setSize(400, 600);
        f.setContentPane(new ShuttlePanel());
        f.setVisible(true);
    }
}
```

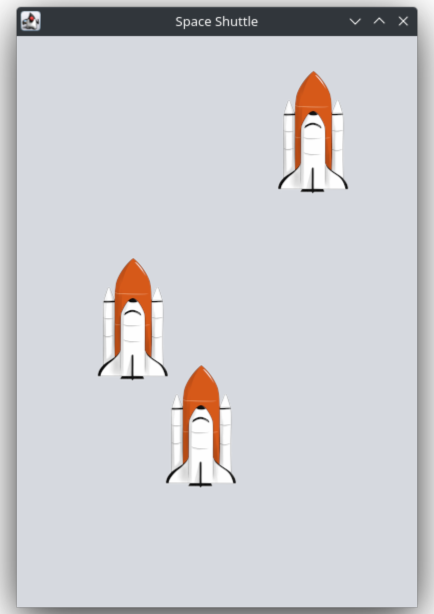
`shuttle.png`



The breakpoints (*breakpoints* in English) are marks that indicate in the IDE where the execution of the code should stop when it is done in debug mode.

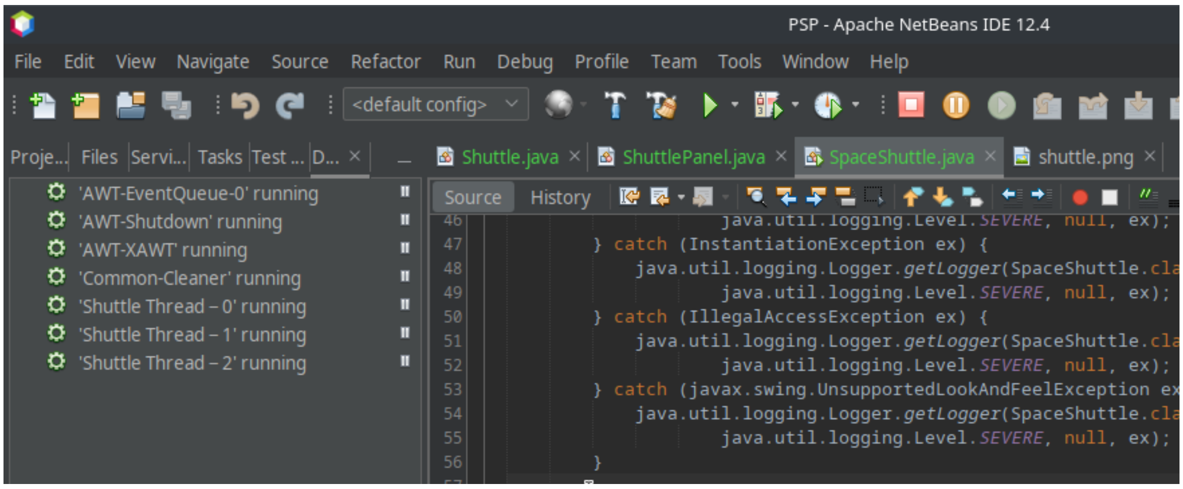
If no breakpoints are set, the application will run to completion. The application is a panel that shows images that move. Each image (`shuttle`) is controlled by a thread. The application creates 3 threads, so there will be three images in the moving panel.

Application of `Space shuttle` in operation:



In the debug window we can see the three threads created, named `Shuttle Thread 0`, `Shuttle Thread 1` and Shuttle Thread 2. The rest of the threads created are the main and system threads.

Thread debugging:

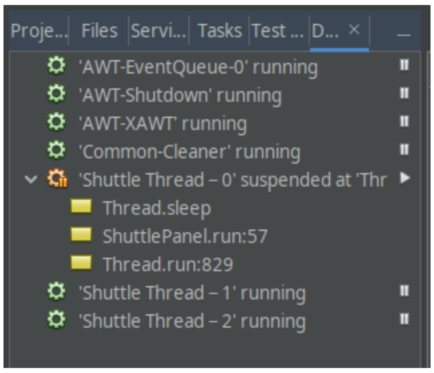

PSP - Apache NetBeans IDE 12.4
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Proje... Files Servi... Tasks Test ... D...
Shuttle.java
ShuttlePanel.java
SpaceShuttle.java
shuttle.png
Source History

```
'AWT-EventQueue-0' running
'AWT-Shutdown' running
'AWT-XAWT' running
'Common-Cleaner' running
'Shuttle Thread – 0' running
'Shuttle Thread – 1' running
'Shuttle Thread – 2' running
```

```
                java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {
            java.util.logging.Logger.getLogger(SpaceShuttle.cla
                java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(SpaceShuttle.cla
                java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex
            java.util.logging.Logger.getLogger(SpaceShuttle.cla
                java.util.logging.Level.SEVERE, null, ex);
        }
```

To the right of the thread name is a pause symbol. If we click it the thread will stop. The symbol will become the *play* triangle. When the thread is stopped we can see that an expandable folder comes out. Inside we see the list of thread calls. Since the thread is stopped, the shuttle (the image moving around the panel) stops too.

Stopped thread:



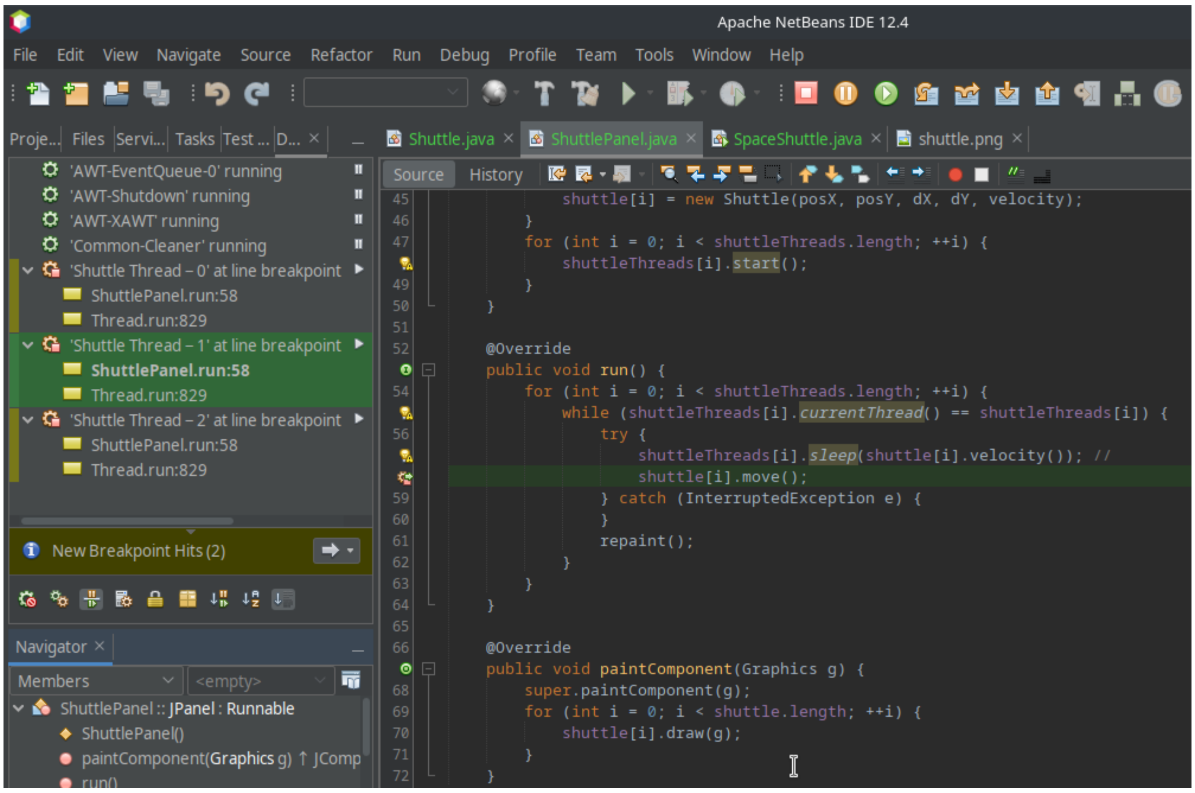
```
'AWT-EventQueue-0' running
'AWT-Shutdown' running
'AWT-XAWT' running
'Common-Cleaner' running
'Shuttle Thread – 0' suspended at 'Thr
    Thread.sleep
    ShuttlePanel.run:57
    Thread.run:829
'Shuttle Thread – 1' running
'Shuttle Thread – 2' running
```

If we have put a breakpoint, the application behaves as if it were not working with threads. Execution stops at this point.

To put a breakpoint we must click in the left margin of the editor next to the line number in which we want to put the breakpoint. A red square will appear marking the point.

In the figure below you can see that the breakpoint is set when the thread executes the image move.

Break point:



```
'AWT-EventQueue-0' running
'AWT-Shutdown' running
'AWT-XAWT' running
'Common-Cleaner' running
'Shuttle Thread – 0' at line breakpoint
    ShuttlePanel.run:58
    Thread.run:829
'Shuttle Thread – 1' at line breakpoint
    ShuttlePanel.run:58
    Thread.run:829
'Shuttle Thread – 2' at line breakpoint
    ShuttlePanel.run:58
    Thread.run:829
```

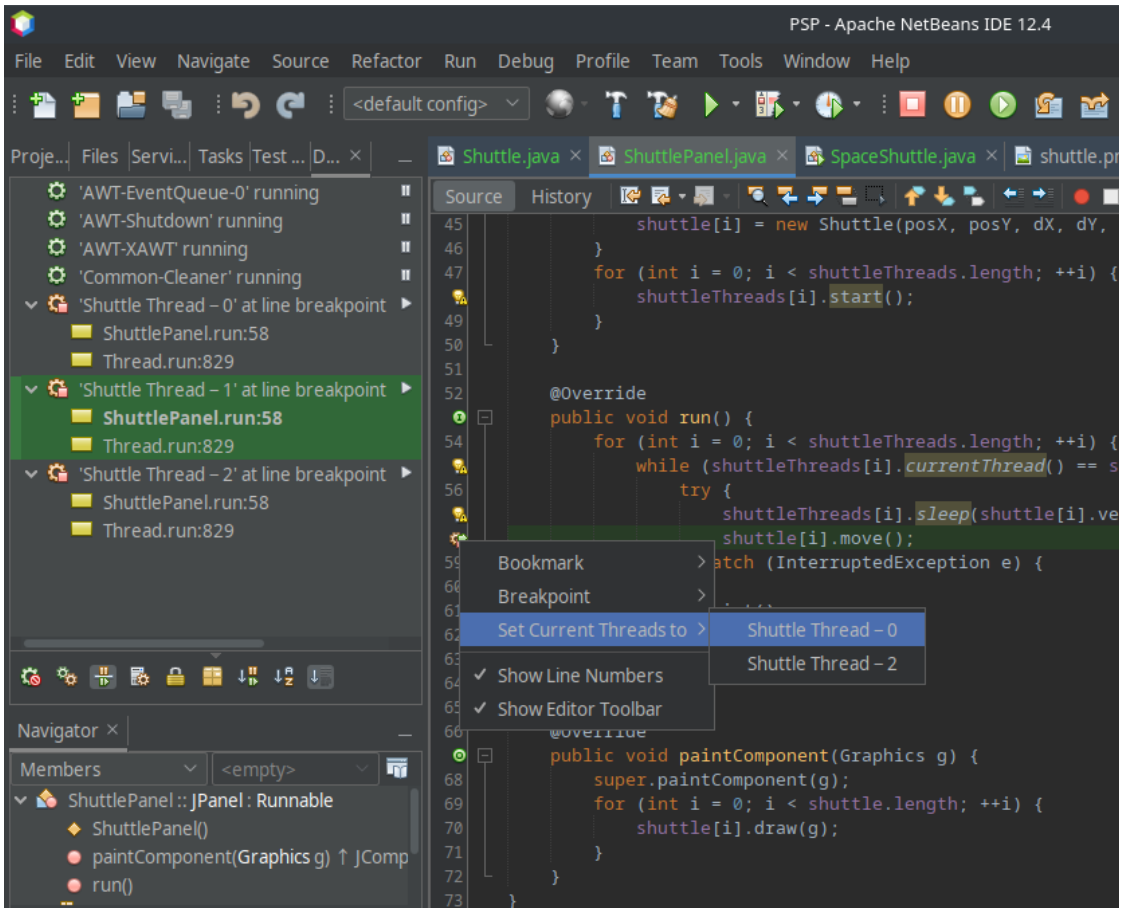
```
ShuttlePanel :: JPanel : Runnable
    ShuttlePanel()
    paintComponent(Graphics g) ↑ JComp
    run()
```

```java
            shuttle[i] = new Shuttle(posX, posY, dX, dY, velocity);
        }
        for (int i = 0; i < shuttleThreads.length; ++i) {
            shuttleThreads[i].start();
        }
    }

    @Override
    public void run() {
        for (int i = 0; i < shuttleThreads.length; ++i) {
            while (shuttleThreads[i].currentThread() == shuttleThreads[i]) {
                try {
                    shuttleThreads[i].sleep(shuttle[i].velocity()); //
                    shuttle[i].move();
                } catch (InterruptedException e) {
                }
                repaint();
            }
        }
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        for (int i = 0; i < shuttle.length; ++i) {
            shuttle[i].draw(g);
        }
    }
```

In the debugging window you can see how the first thread that arrives is being executed. Pressing the `F7` or `F8` function keys will advance the program line on the currently active thread. The rest of the threads are ready to execute. If we want the other threads to start, we must click on the arrow below the debug window and select the thread that we want to execute. Once all the threads are running, we can see the execution of one or another thread by clicking on the thread in the debug window. The thread we control appears with a green frame.

To the left of the code there are gear icons that indicate where the threads that we are not debugging are stopped. If we click on the toothed wheel with the right button we can see which thread is stopped at this point.

Swapping debug threads:


PSP - Apache NetBeans IDE 12.4
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Proje... Files Servi... Tasks Test ... D...
Shuttle.java ShuttlePanel.java SpaceShuttle.java shuttle.pr
Source History
Bookmark
Breakpoint
Set Current Threads to
Shuttle Thread – 0
Shuttle Thread – 2
Show Line Numbers
Show Editor Toolbar
Navigator
Members
<empty>
ShuttlePanel :: JPanel : Runnable
ShuttlePanel()
paintComponent(Graphics g) ↑ JComp
run()

```
'AWT-EventQueue-0' running
'AWT-Shutdown' running
'AWT-XAWT' running
'Common-Cleaner' running
'Shuttle Thread – 0' at line breakpoint
    ShuttlePanel.run:58
    Thread.run:829
'Shuttle Thread – 1' at line breakpoint
    ShuttlePanel.run:58
    Thread.run:829
'Shuttle Thread – 2' at line breakpoint
    ShuttlePanel.run:58
    Thread.run:829
```

```java
                shuttle[i] = new Shuttle(posX, posY, dX, dY,
            }
            for (int i = 0; i < shuttleThreads.length; ++i) {
                shuttleThreads[i].start();
            }
        }

        @Override
        public void run() {
            for (int i = 0; i < shuttleThreads.length; ++i) {
                while (shuttleThreads[i].currentThread() == s
                    try {
                        shuttleThreads[i].sleep(shuttle[i].ve
                        shuttle[i].move();
                    atch (InterruptedException e) {



        public void paintComponent(Graphics g) {
            super.paintComponent(g);
            for (int i = 0; i < shuttle.length; ++i) {
                shuttle[i].draw(g);
            }
        }
    }
```

If we want to switch to continue debugging another thread we will select it from the list. We can only debug a thread with a breakpoint. When we click on `F7` or `F8` to move forward only one thread advances, the others do not stop at the breakpoint.

2. Looking for deadlocks

Debugging applications is quite simple and very useful. IDEs increasingly provide tools that help us more to correct errors. Netbeans in its debugging tool includes a feature that detects deadlocks automatically. We will use the following code snippet.

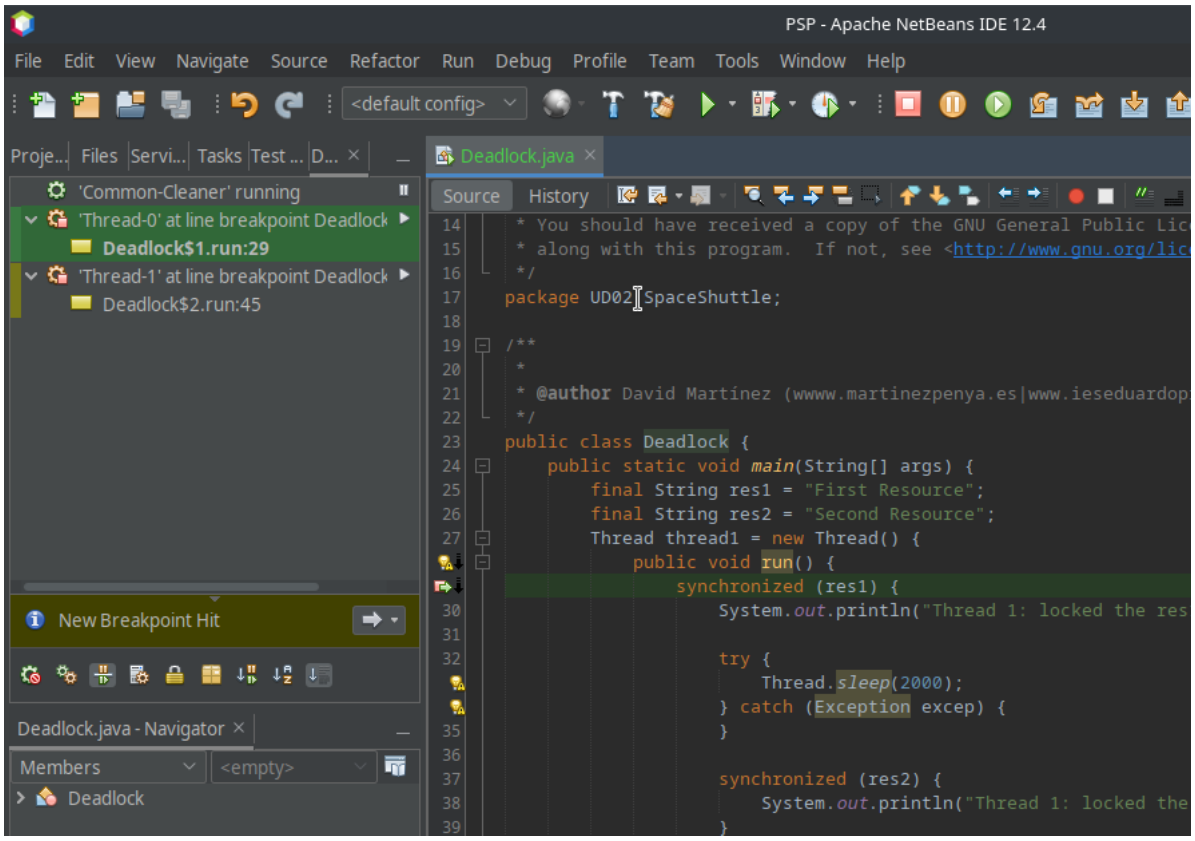
`Deadlock.java`

```java
 1  public class Deadlock {
 2      public static void main(String[] args) {
 3          final String res1 = "First Resource";
 4          final String res2 = "Second Resource";
 5          Thread thread1 = new Thread() {
 6              public void run() {
 7                  synchronized (res1) {
 8                      System.out.println("Thread 1: locked the res1");
 9  
10                      try {
11                          Thread.sleep(2000);
12                      } catch (Exception excep) {
13                      }
14  
15                      synchronized (res2) {
16                          System.out.println("Thread 1: locked the res2");
17                      }
18                  }
19              }
20          };
21          Thread thread2 = new Thread() {
22              public void run() {
23                  synchronized (res2) {
24                      System.out.println("Thread 2: locked the res2");
25  
26                      try {
27                          Thread.sleep(2000);
28                      } catch (Exception excep) {
29                      }
30  
31                      synchronized (res1) {
32                          System.out.println("Thread 2: locked the res1");
33                      }
34                 }
35              }
36          };
37          thread1.start();
38          thread2.start();
39      }
40  }
```

To use it we must create breakpoints. Breakpoints are placed where we think the critical point of our application exists and where it could be the source of errors. In this case we'll put them in the `run()` method of the class that inherits `threads` or implements `runnable`. In this case on lines 7 and 23.

We run the app in debug mode and see how it stops at breakpoints.

Break point:

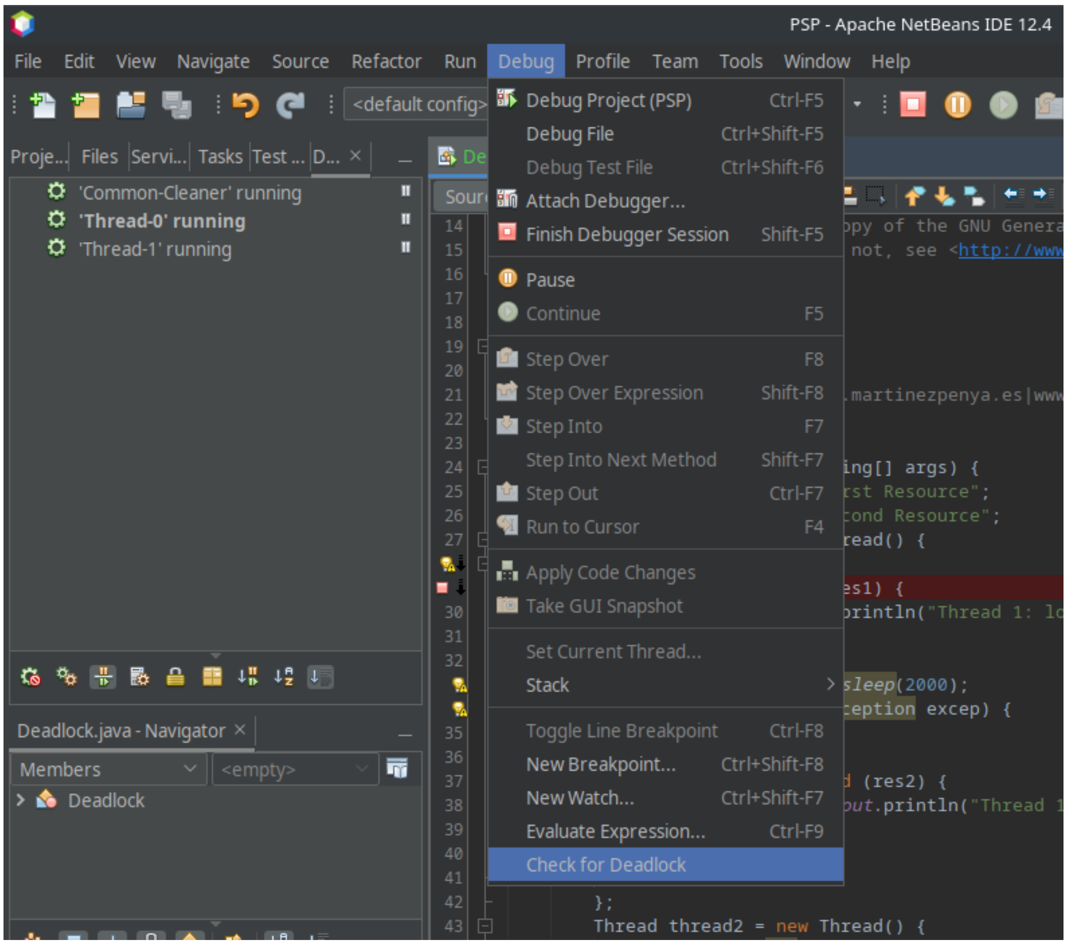

PSP - Apache NetBeans IDE 12.4
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Proje... Files Servi... Tasks Test ... D...
'Common-Cleaner' running
'Thread-0' at line breakpoint Deadlock
Deadlock$1.run:29
'Thread-1' at line breakpoint Deadlock
Deadlock$2.run:45
New Breakpoint Hit
Deadlock.java - Navigator
Members
<empty>
Deadlock
Deadlock.java
Source History

```
 * You should have received a copy of the GNU General Public Lic
 * along with this program.  If not, see <http://www.gnu.org/lic
 */
package UD02.SpaceShuttle;

/**
 *
 * @author David Martínez (wwww.martinezpenya.es|www.ieseduardop
 */
public class Deadlock {
    public static void main(String[] args) {
        final String res1 = "First Resource";
        final String res2 = "Second Resource";
        Thread thread1 = new Thread() {
            public void run() {
                synchronized (res1) {
                    System.out.println("Thread 1: locked the res

                    try {
                        Thread.sleep(2000);
                    } catch (Exception excep) {
                    }

                    synchronized (res2) {
                        System.out.println("Thread 1: locked the
                    }
```

At this point we can continue the execution (function keys `F7` or F8`)` and see how the application progresses and changes the values of the variables, the line of execution, etc.

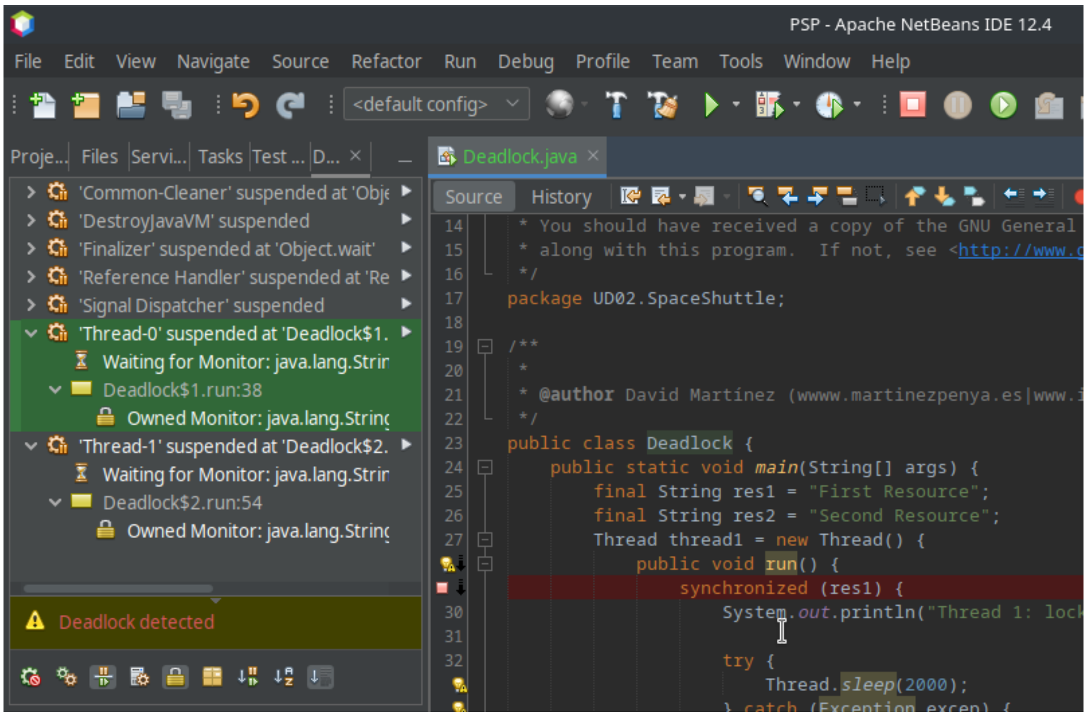
To detect the deadlock we must terminate the execution. We can click on each right button and select `resume`, we can press `F5` or click on the white triangle toolbar icon (*play*) on a green background. The result we will have is the application running. To see if a deadlock has occurred we must go to the debug menu and click on the `Check for DeadLock` option.

Examine deadlock:


PSP - Apache NetBeans IDE 12.4
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Proje... Files Servi... Tasks Test ... D...
'Common-Cleaner' running
'Thread-0' running
'Thread-1' running
Deadlock.java - Navigator
Members
<empty>
Deadlock
Debug Project (PSP) Ctrl-F5
Debug File Ctrl+Shift-F5
Debug Test File Ctrl+Shift-F6
Attach Debugger...
Finish Debugger Session Shift-F5
Pause
Continue F5
Step Over F8
Step Over Expression Shift-F8
Step Into F7
Step Into Next Method Shift-F7
Step Out Ctrl-F7
Run to Cursor F4
Apply Code Changes
Take GUI Snapshot
Set Current Thread...
Stack
Toggle Line Breakpoint Ctrl-F8
New Breakpoint... Ctrl+Shift-F8
New Watch... Ctrl+Shift-F7
Evaluate Expression... Ctrl-F9
Check for Deadlock

Netbeans will automatically tell us if there are deadlocks or not. It will show us a message saying if it has found it or not.

Interlock message:


PSP - Apache NetBeans IDE 12.4
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Proje... Files Servi... Tasks Test ... D...
Deadlock.java
Source History

```
'Common-Cleaner' suspended at 'Obje
'DestroyJavaVM' suspended
'Finalizer' suspended at 'Object.wait'
'Reference Handler' suspended at 'Re
'Signal Dispatcher' suspended
'Thread-0' suspended at 'Deadlock$1.
    Waiting for Monitor: java.lang.Strir
    Deadlock$1.run:38
        Owned Monitor: java.lang.Strinc
'Thread-1' suspended at 'Deadlock$2.
    Waiting for Monitor: java.lang.Strir
    Deadlock$2.run:54
        Owned Monitor: java.lang.Strinc
Deadlock detected
```

```java
 * You should have received a copy of the GNU General
 * along with this program.  If not, see <http://www.g
 */
package UD02.SpaceShuttle;

/**
 *
 * @author David Martínez (wwww.martinezpenya.es|www.i
 */
public class Deadlock {
    public static void main(String[] args) {
        final String res1 = "First Resource";
        final String res2 = "Second Resource";
        Thread thread1 = new Thread() {
            public void run() {
                synchronized (res1) {
                    System.out.println("Thread 1: lock

                    try {
                        Thread.sleep(2000);
                    } catch (Exception excep) {
```

The debug window tells us where the deadlocked threads are.

Debugging tools are very useful for finding possible programming bugs. When programming using many threads, application debugging is essential.

3. Information sources

- Wikipedia
- Programación de servicios y procesos - FERNANDO PANIAGUA MARTÍN [Paraninfo]
- Programación de Servicios y Procesos - ALBERTO SÁNCHEZ CAMPOS [Ra-ma]
- Programación de Servicios y Procesos - Mª JESÚS RAMOS MARTÍN - [Garceta] (1ª y 2ª Edición)
- Programación de servicios y procesos - CARLOS ALBERTO CORTIJO BON [Sintesis]
- Programació de serveis i processos - JOAR ARNEDO MORENO, JOSEP CAÑELLAS BORNAS i JOSÉ ANTONIO LEO MEGÍAS [IOC]
- GitHub repositories:
  - https://github.com/ajcpro/psp
  - https://oscarmaestre.github.io/servicios/index.html
  - https://github.com/juanro49/DAM/tree/master/DAM2/PSP
  - https://github.com/pablohs1986/dam_psp2021
  - https://github.com/Perju/DAM
  - https://github.com/eldiegoch/DAM
  - https://github.com/eldiegoch/2dam-psp-public
  - https://github.com/franlu/DAM-PSP
  - https://github.com/ProgProcesosYServicios
  - https://github.com/joseluisgs
  - https://github.com/oscarnovillo/dam2_2122
  - https://github.com/PacoPortillo/DAM_PSP_Tarea02_La-Cena-de-los-Filosofos